Основные J2EE-паттерны
Session Facade (фасад сессии)

Контекст

Корпоративные компоненты инкапсулируют бизнес-логику и бизнес-данные и предоставляют их интерфейсы, а следовательно и сложность распределенных служб, клиентскому уровню.

Проблема

В многоуровневой прикладной среде Java 2 Platform, Enterprise Edition (J2EE) возникают следующие проблемы:

- Тесные связи, ведущие к прямой взаимной зависимости клиентов и бизнес-объектов.
- Слишком большое количество вызовов методов между клиентом и сервером вызывает снижение сетевой производительности.
- Отсутствие унифицированной стратегии клиентского доступа порождает некорректное использование бизнес-объектов.

Многоуровневое J2EE-приложение применяет многочисленные серверные объекты, реализуемые как корпоративные компоненты. Кроме того, некоторые из вспомогательных объектов могут предоставлять службы, данные или и то и другое. Все эти объекты называются бизнес-объектами, поскольку инкапсулируют бизнес-данные и бизнес-логику.

J2EE-приложения реализуют бизнес-объекты, обеспечивающие функционирование служб, в виде сессионных компонентов. Бизнес-объекты общего назначения, представляющие функции постоянного хранения с объектной стороны и используемые несколькими пользователями, обычно реализуются в виде компонентов управления данными.

Клиенты приложения должны получить доступ к бизнес-объектам для выполнения своих функций и для удовлетворения требований пользователя. Клиенты могут непосредственно взаимодействовать с этими бизнес-объектами, поскольку те предоставляют свои интерфейсы. При предоставлении бизнес-объектов клиенту, он должен понимать и нести ответственность за объектные взаимосвязи бизнес-данных и должен быть способен контролировать последовательность бизнес-процессов.

Однако, прямое взаимодействие клиента и бизнес-объектов приводит к тесным связям между ними и эти связи ставят клиента в прямую зависимость от реализации бизнес-объектов. Прямая зависимость означает, что клиент должен предоставить и реализовать сложные взаимодействия, касающиеся процессов поиска и создания бизнес-объекта, управлять взаимосвязями между

принимающими участие в этих процессах бизнес-объектами, а также нести ответственность за разграничение транзакций.

По мере увеличения требований клиента сложность взаимодействия между различными бизнес-объектами растет. Для удовлетворения этих требований клиентское приложение увеличивается и становится более сложным. Оно становится также очень чувствительным к изменениям на уровне бизнес-объектов и, кроме того, клиент не всегда соответствует сложности системы, в которой работает.

Тесная связь между объектами дает о себе знать и при управлении связями объектов между собой. Часто не ясно, где осуществляется управление связями. Это приводит к сложным взаимосвязям между бизнес-объектами и потере гибкости приложения. Отсутствие гибкости делает приложение менее управляемым при необходимости изменений.

При обращении к корпоративным компонентам клиенты взаимодействуют с удаленными объектами. Если клиент непосредственно взаимодействует со всеми участвующими бизнес-объектами, могут возникнуть проблемы сетевой производительности. Каждый вызов клиентом корпоративного компонента потенциально является удаленным вызовом метода. Каждое обращение к бизнес-объекту занимает относительно немного ресурсов. По мере роста количества участников в сценарии, число таких удаленных вызовов методов увеличивается. При увеличении числа удаленных вызовов методов увеличивается трафик между клиентом и серверными бизнес-объектами. Это может повлечь уменьшение сетевой производительности приложения, поскольку большой объем удаленных вызовов методов увеличивает количество взаимодействий на сетевом уровне.

Проблема также возникает при непосредственном взаимодействии клиента с бизнес-объектами. Поскольку бизнес-объекты представлены клиентам явно, не существует унифицированной стратегии получения доступа к ним. Без такой унификации бизнес-объекты представляются клиентам, что может ослабить непротиворечивое их использование.

Ограничения

- Обеспечить более простой интерфейс для клиентов посредством сокрытия всех сложных взаимодействий между бизнес-компонентами.
- Уменьшить количество бизнес-объектов, представленных клиенту по сети уровнем служб.
- Скрыть от клиента взаимодействия и взаимозависимости между бизнес-компонентами, на которых основывается их функционирование. Это обеспечивает лучшую управляемость, централизацию взаимодействий (надежность), большую гибкость и большую приспособленность к изменениям.
- Обеспечить унифицированный уровень служб для отделения реализации бизнес-объекта от абстракции бизнес-службы.
- Избежать предоставления низкоуровневых бизнес-объектов непосредственно

клиенту для сведения к минимуму тесной связи между этими двумя уровнями.

Решение

Используйте сессионный компонент в качестве «фасада» для инкапсулирования сложности взаимодействий между участвующими в рабочем процессе бизнес-объектами. Session Facade управляет бизнес-объектами и обеспечивает унифицированный общий уровень доступа к службам для клиентов.

Session Facade абстрагирует взаимодействия бизнес-объектов и обеспечивает уровень служб, предоставляющий только необходимые интерфейсы. Таким образом, он скрывает со стороны клиента сложные взаимодействия между участниками этих взаимодействий. Session Facade управляет взаимодействиями между объектами бизнес-данных и бизнес-служб, участвующих в рабочем процессе, и инкапсулирует связанную с этим бизнес-логику. Другими словами, сессионный компонент (представляющий Session Facade) управляет взаимосвязями между бизнес-объектами. Сессионный компонент управляет также жизненным циклом этих участников посредством создания, поиска, изменения и удаления их в течение рабочего цикла по необходимости. В сложных приложениях Session Facade может передать управление жизненным циклом отдельному объекту. Например, для управления циклом жизни сессионного компонента и компонента управления данными Session Facade может передать соответствующие функции объекту Service Locator (см. раздел "Service Locator" на стр. 368).

Важно рассмотреть взаимоотношения между бизнес-объектами. Некоторые взаимоотношения между бизнес-объектами являются транзитными, то есть используемыми только в конкретном отношении или сценарии. Другие взаимоотношения могут быть более постоянными. Транзитные взаимоотношения в Session Facade лучше моделируются в виде рабочего процесса, в котором Session Facade управляет отношениями между бизнес-объектами. Постоянные взаимоотношения между двумя бизнес-объектами должны быть изучены для определения того, какой бизнес-объект (если не оба) управляет этим взаимоотношением.

Функции и объекты Session Facade

Итак, как можно определить объекты Session Facade при моделировании функций? Отображение каждой функции на Session Facade приведет к слишком большому количеству объектов Session Facade. Это сведет на нет наши намерения использовать наименьшее количество сессионных компонентов общего назначения. Вместо этого, после получения Session Facade при моделировании, объедините их в небольшое количество сессионных компонентов, основываясь на некоторой логической декомпозиции.

Например, для банковского приложения вы можете сгруппировать взаимодействия, относящиеся к управлению счетом, в один компонент. Функции

«Создать новый счет», «Изменить информацию о счете», «Просмотр информации о счете» и т.д. имеют дело с объектом управления данными общего назначения Account. Создание компонента Session Facade для каждой функции не рекомендуется. То есть, функции, требуемые для поддержки этих логически связанных задач, могут быть сгруппированы вместе в одном Session Facade называемом AccountSessionFacade.

В данном случае Session Facade будет выступать в качестве контроллера самого общего назначения с высокоуровневыми методами, которые могут обрабатывать каждое взаимодействие (то есть, createNewAccount, changeAccount, getAccount). Мы рекомендуем использовать Session Facade для группировки связанных взаимодействий в одном Session Facade. Это приведет к уменьшению объектов Session Facade в приложении и даст возможность использовать преимущества паттерна Session Facade.

Структура

На рисунке 8.15 показана диаграмма классов, представляющих паттерн Session Facade.

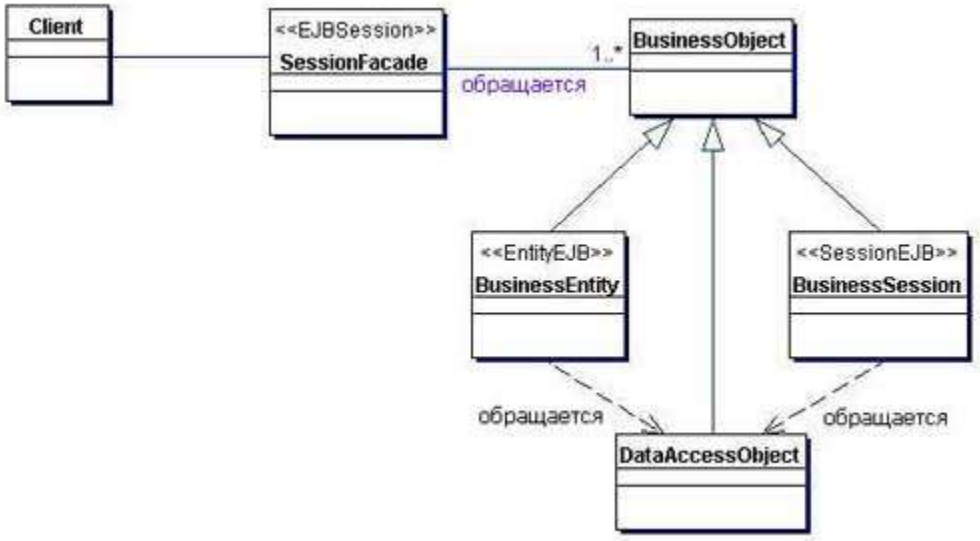

Рисунок 8.15 Диаграмма классов Session Facade

Участники и обязанности

Рисунок 8.16 содержит диаграмму последовательности действий, показывающую взаимодействия Session Facade с двумя компонентами управления данными, одним сессионным компонентом и объектом DAO. Все они принимают участие в выполнении запроса от клиента.

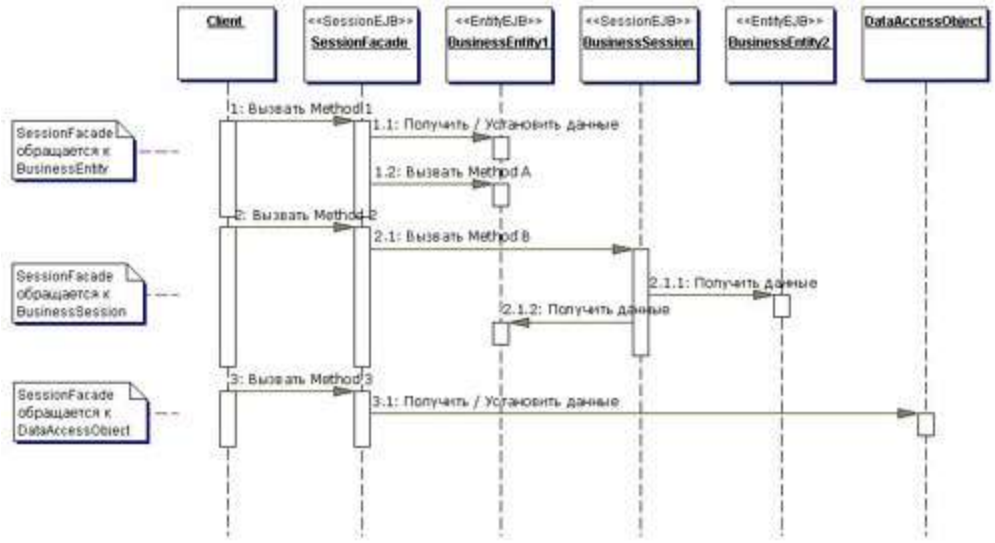

Рисунок 8.16 Диаграмма последовательности действий Session Facade

Client

Представляет клиента Session Facade, которому необходим доступ к бизнес-службе. Этим клиентом может быть другой сессионный компонент (Session Facade) в том же самом бизнес-уровне, либо Business Delegate (см. раздел "Business Delegate" на стр. 248) в другом уровне.

SessionFacade

SessionFacade реализуется как сессионный компонент. SessionFacade управляет взаимосвязями между многочисленными объектами BusinessObject и обеспечивает более высокий уровень абстракции для клиента. SessionFacade предлагает общий доступ к задействованному BusinessObject, представленному при помощи вызова Invoke сессионному компоненту.

BusinessObject

BusinessObject представляет собой ролевой объект, способствующий применению различных стратегий для сессионных компонентов, компонентов управления данными и DAO (см. следующий раздел “Стратегии”). BusinessObject предоставляет данные и/или службу в диаграмме классов. SessionFacade взаимодействует с несколькими экземплярами BusinessObject.

Стратегии

Session Facade представляет собой объект контроллера уровня служб, который управляет взаимодействиями между клиентом и задействованными объектами бизнес-данных и бизнес-служб. Вы можете определить, где может быть полезным применение Session Facade, изучив клиентские требования и взаимодействия, которые обычно описаны в функциях и сценариях. Этот анализ даст вам возможность определить уровень контроллера (заключаемого в Session Facade), который может выступать как фасад для этих сценариев.

В этом разделе объясняются различные стратегии для реализации Session Facade.

Стратегии Session Facade

Стратегия Stateless Session Facade

При реализации Session Facade прежде всего вы должны решить, является ли компонент Session Facade сессионным компонентом, имеющим состояние, либо компонентом без состояния. Принимайте это решение в зависимости от бизнес-процессов, которые моделируются в Session Facade.

Бизнес-процесс, требующий только одного вызова метода для завершения службы, называется бездиалоговым бизнес-процессом. Такие процессы хорошо моделируются сессионным компонентом, не имеющим состояния.

Тщательное изучение функций и сценариев даст вам возможность указать определения Session Facade. Если функция является бездиалоговой, то клиент инициирует эту функцию, используя единственный метод в Session Facade. После завершения работы метода функция также завершается. Нет необходимости хранить состояние диалога между одним вызовом метода и следующим. В этом сценарии Session Facade может быть реализован в виде сессионного компонента, не имеющего состояния.

Стратегия Stateful Session Facade

Бизнес-процесс, которому необходимо выполнить несколько вызовов метода для завершения службы, называется диалоговым. Состояние диалога должно быть сохранено между каждым вызовом метода клиентом. В этом сценарии для реализации Session Facade более подходит применение сессионного компонента, сохраняющего состояние.

В обеих стратегиях (и в Stateless Session Facade, и в Stateful Session Facade) роль бизнес-объекта может быть выполнена различными способами. Они рассмотрены ниже.

Стратегии бизнес-объектов

Вы можете реализовать бизнес-объект в виде сессионного компонента, компонента управления данными, объекта DAO или обычного Java-объекта. В следующих стратегиях обсуждается каждый из этих вариантов.

Стратегия сессионного компонента

Бизнес-объект может быть реализован как сессионный компонент. Сессионный компонент обычно предоставляет бизнес-службу и, в некоторых случаях, может также предоставлять бизнес-данные. Когда такой сессионный компонент нуждается в получении доступа к данным, он может использовать DAO для работы с ними. Session Facade может содержать один или более таких ориентированных на

службы либо на данные сессионных компонентов, действующих как бизнес-объекты.

Стратегия компонента управления данными

Представление бизнес-объекта при помощи компонента управления данными является наиболее типичным использованием Session Facade. Когда несколько таких компонентов участвуют в функции, нет необходимости предоставлять клиенту все компоненты управления данными. Вместо этого Session Facade может заключить в себя эти компоненты и предоставить метод общего назначения для выполнения требуемой бизнес-функции, скрывая, таким образом, сложные взаимодействия компонентов.

Стратегия Data Access Object

Session Facade может непосредственно использовать один или более объектов DAO для представления бизнес-данных. Такая стратегия используется тогда, когда приложение является настолько простым, что не требует компонентов управления данными, либо архитектура приложения базируется только на сессионных компонентах и не использует компоненты управления данными. Использование DAO внутри сессионных компонентов частично симулирует персистентную природу компонентов управления данными.

Приложение может нуждаться в службах, предоставляемых произвольным Java-объектом (то есть, объектом, не являющимся корпоративным компонентном или объектом DAO, хотя DAO может рассматриваться как тип произвольного Java-объекта). В этих случаях Session Facade обращается к этому Java-объекту для обеспечения необходимой функциональности.

Выводы

- **Вводит бизнес-уровень контроллера**
  Session Facade может представлять промежуточное звено между клиентом и бизнес-уровнем, как определено в процессе исследовательского моделирования. Session Facade осуществляет взаимодействия между клиентом и бизнес-компонентами. В сложном приложении вы можете определить несколько Session Facade, которые могут быть посредниками между клиентом и участвующими объектами бизнес-уровня. Для более простых приложений можно предположить, что Session Facade не представляет большой ценности, поскольку он может выступать в основном как прокси для клиентских запросов к одиночному бизнес-компоненту. Однако, по мере увеличения сложности приложения, использование Session Facade принесет преимущества на более поздних стадиях разработки.
- **Предоставляет унифицированный интерфейс**
  Лежащие в основе функционирования бизнес-компонентов взаимодействия могут быть очень сложными. Паттерн Session Facade абстрагирует эту сложность и предоставляет клиенту простой интерфейс, более легкий для понимания и использования. Применяя Session Facade, вы можете разработать уровень служб,

предоставляющий более простые интерфейсы к системе в целом. То есть, Session Facade предоставляет унифицированный уровень доступа общего назначения клиентам всех типов и может защитить и скрыть задействованные бизнес-компоненты.

- **Уменьшает связность, увеличивает управляемость**
  Использование Session Facade отделяет бизнес-объекты от клиентов, уменьшая при этом тесную связь и зависимость клиентов от бизнес-компонентов. Наилучшим решением является применение Session Facade для управления рабочим процессом среди бизнес-объектов, а не проектирование бизнес-объектов, уведомляющих друг друга. Бизнес-объект должен отвечать только за свои собственные действия (данные и логику). Взаимодействие между бизнес-объектами может быть абстрагировано в Session Facade. Это обеспечит лучшую управляемость, централизацию взаимодействий (надежность), большую гибкость и большую приспособленность к изменениям.
  Выделение рабочего процесса в Session Facade устраняет прямую зависимость клиента от участвующих в этом процессе объектов и придает гибкость разработке. Хотя изменения в объектах-участниках может потребовать изменений в Session Facade, централизация в нем рабочего процесса делает такие изменения более управляемыми. Нужно изменить только Session Facade, а не каждый из клиентов. Код клиента упрощается, поскольку он передает ответственность за рабочий цикл в Session Facade. Клиент более не управляет сложными взаимодействиями между бизнес-объектами и не беспокоится о взаимозависимостях между ними.
- **Улучшает производительность, уменьшает число специализированных методов**
  Session Facade также влияет на производительность. Session Facade уменьшает сетевые накладные расходы между клиентом и сервером, поскольку его использование устраняет прямое взаимодействие между клиентом и объектами бизнес-данных и бизнес-служб. Все взаимодействия перенаправляются в Session Facade. Session Facade и участники процесса находятся ближе друг к другу, что увеличивает эффективность управления взаимодействиями между объектами-участниками. Вся передача данных и все вызовы методов от Session Facade к участникам предположительно осуществляются в относительно скоростной сети. Производительность сети может быть еще улучшена для обеспечения максимальной пропускной способности при использовании, по возможности, паттерна Transfer Object для объектов-участников.
- **Обеспечивает доступ общего назначения**
  Session Facade представляет собой очень общую абстракцию рабочего процесса. Таким образом, нежелательно иметь один Session Facade на одно взаимодействие компонента управления данными, который может предоставить более точную абстракцию. Проанализируйте взаимодействие между клиентом и службами приложения, используя функции и сценарии для определения степени “грубости” паттерна. Определите оптимальную модульность Session Facade для приложения путем разделения приложения на логические подсистемы и предоставьте Session Facade для каждой подсистемы. Однако, предоставление одного Session Facade для всей системы может привести к очень большому размеру Session Facade, чьи многочисленные методы могут сделать его

использование не эффективным. Одиночного Session Facade может быть достаточно только для очень простых приложений, не имеющих подсистем.

- **Централизует управление безопасностью**
  Политика безопасности приложения может управляться на уровне Session Facade, поскольку он является уровнем, представленным клиенту. Из-за общего доступа к Session Facade легче и более осуществимо определять политики безопасности на этом уровне, а не на уровне участвующих бизнес-компонентов. Бизнес-компоненты предлагают специализированное управление, тогда как легче управлять безопасностью в предоставляющем обобщенный доступ Session Facade, поскольку существует относительно небольшое число методов общего назначения.
- **Централизует управление транзакциями**
  Поскольку Session Facade реализует рабочий процесс функций, более логично разместить управление транзакциями на уровне Session Facade. Централизованное управление транзакциями обладает такими же преимуществами, что и централизованное управление безопасностью. Session Facade предлагает центральное место для управления и определения транзакций. Для реализации управления транзакциями в бизнес-компонентах необходимо выполнить намного больше работы, поскольку они более специализированы. Кроме того, при прямом обращении клиента в корпоративный компонент минуя Session Facade, груз разделения транзакций ложится на клиента и может привести к нежелательным результатам.
- **Предоставляет клиентам меньше удаленных интерфейсов**
  Клиенты, взаимодействующие непосредственно с объектами бизнес-данных и бизнес-служб, вызывают увеличение трафика между клиентом и сервером. Это может привести к уменьшению сетевой производительности. Все обращения к бизнес-объекту должны осуществляться через более высокий уровень абстракции, представленный в Session Facade. Поскольку он предоставляет более общий механизм доступа к бизнес-компонентам, уменьшается количество бизнес-компонентов, представленных клиенту. Следовательно, пределы уменьшения производительности понижаются из-за ограниченного числа взаимодействий между клиентами и Session Facade по сравнению с прямым взаимодействием клиента и отдельных бизнес-компонентов.

Пример

Реализация Session Facade

Рассмотрим приложение Professional Services Application (PSA), в котором рабочий процесс, относящийся к компонентам управления данными (например, Project, Resource), инкапсулирован в ProjectResourceManagerSession, реализованном с использованием паттерна Session Facade. В примере 8.15 показано взаимодействие с компонентами Resource и Project, а также с другими бизнес-компонентами, например Value List Handlers (см. раздел "Value List Handler" на стр. 354) и Transfer Object Assemblers (см. раздел "Transfer Object Assembler" на стр. 340).

Пример 8.15 Реализация Session Facade - сессионный компонент

```
package corepatterns.apps.psa.ejb;

import java.util.*;
import java.rmi.RemoteException;
import javax.ejb.*;
import javax.naming.*;
import corepatterns.apps.psa.core.*;
import corepatterns.util.ServiceLocator;
import corepatterns.util.ServiceLocatorException;

// Обратите внимание: для краткости детали try/catch не показаны.

public class ProjectResourceManagerSession
  implements SessionBean {

  private SessionContext context;

  // Удаленные ссылки для компонентов
  // управления данными, инкапсулированными в этом фасаде
 private Resource resourceEntity = null;
  private Project projectEntity = null;
  ...

 // создание по умолчанию
  public void ejbCreate()
  throws CreateException {
  }

  // метод для создания этого фасада и
 // установления соединений к необходимым
  // компонентам управления данными
  // при помощи значений первичного ключа
 public void ejbCreate(
    String resourceId, String projectId, ...)
  throws CreateException, ResourceException {

    try {
   // найти и создать соединение с компонентами
      connectToEntities(resourceId, projectId, ...);
    } catch(...) {
   // Обработка исключительных ситуаций
    }
  }

 // метод для получения соединения Session Facade
  // с его компонентами управления данными при помощи
 // значений первичного ключа
  private void connectToEntities (
    String resourceId, String projectId)
  throws ResourceException {
    resourceEntity = getResourceEntity(resourceId);
    projectEntity = getProjectEntity(projectId);
    ...
  }

 // метод для получения соединения Session Facade
```

```
 // с различным набором компонентов управления данными
// при помощи значений первичного ключа
 public resetEntities(String resourceId,
   String projectId, ...)
 throws PSAException {

   connectToEntities(resourceId, projectId, ...);
 }

// private-метод для получения Home для Resource
private ResourceHome getResourceHome()
 throws ServiceLocatorException {
   return ServiceLocator.      getInstance().getHome(
       "ResourceEntity", ResourceHome.class);
 }

// private-метод для получения Home для Project
private ProjectHome getProjectHome()
 throws ServiceLocatorException {
   return ServiceLocator.      getInstance().getHome(
       "ProjectEntity", ProjectHome.class);
 }

// private-метод для получения Resource
 private Resource getResourceEntity(
   String resourceId)   throws ResourceException {
   try {
     ResourceHome home = getResourceHome();
     return (Resource)
       home.findByPrimaryKey(resourceId);
   } catch(...) {
  // Обработка исключительных ситуаций
   }
 }

// private-метод для получения Project
 private Project getProjectEntity(String projectId)
 throws ProjectException {
   // similar to getResourceEntity
   ...
 }

// Метод для инкапсуляции рабочего процесса
 // относящегося к назначению ресурса проекту.
// Он имеет дело с компонентами Project и Resource
 public void assignResourceToProject(int numHours)
 throws PSAException {

   try {
     if ((projectEntity == null) ||
         (resourceEntity == null)) {

   // SessionFacade не подключен к сущностям
       throw new PSAException(...);
     }

  // Получить данные Resource
```

```
      ResourceTO resourceTO =
          resourceEntity.getResourceData();

   // Получить данные Project
      ProjectTO projectTO =
        projectEntity.getProjectData();
   // Сначала добавить Resource в Project
      projectEntity.addResource(resourceTO);
   // Создать новый Commitment для Project
      CommitmentTO commitment = new
        CommitmentTO(  ...);

   // добавить commitment в Resource
      projectEntity.addCommitment(commitment);

    } catch(...) {
   // Обработка исключительных ситуаций
    }
  }

  // Подобным образом реализовать другие бизнес-методы для
// обработки различных функций/взаимодействий
  public void unassignResourceFromProject()
  throws PSAException {
    ...
  }

// Методы, работающие с ResourceEntity
  public ResourceTO getResourceData()
  throws ResourceException {
    ...
  }

// Обновить компонент Resource
  public void setResourceData(ResourceTO resource)
  throws ResourceException {
    ...
  }

// Создать новый компонент Resource
public ResourceTO createNewResource(ResourceTO
    resource)   throws ResourceException {
    ...
  }

// Методы для управления временем блокирования ресурса
  public void addBlockoutTime(Collection blockoutTime)
  throws RemoteException,BlockoutTimeException {
    ...
  }

  public void updateBlockoutTime(
    Collection blockoutTime)
  throws RemoteException, BlockoutTimeException {
    ...
  }

  public Collection getResourceCommitments()
```

```
 throws RemoteException, ResourceException {
   ...
 }

// Методы, работающие с ProjectEntity
 public ProjectTO getProjectData()
 throws ProjectException {
   ...
 }

// Обновить компонент Project
 public void setProjectData(ProjectTO project)
 throws ProjectException {
   ...
 }

// Создать новый компонент Project
 public ProjectTO createNewProject(ProjectTO project)
 throws ProjectException {
   ...
 }

 ...

// Другие примеры методов Session Facade

// Служит как прокси вызова к Transfer Object Assembler
// для получения составного Transfer Object.
// См. паттерн Transfer Object Assembler
 public ProjectCTO getProjectDetailsData()
 throws PSAException {
   try {
     ProjectTOAHome projectTOAHome = (ProjectTOAHome)
       ServiceLocator.getInstance().getHome(
         "ProjectTOA", ProjectTOAHome.class);
  // Передать сессионый компонент Transfer Object Assembler
     ProjectTOA projectTOA =
         projectTOAHome.create(...);
     return projectTOA.getData(...);
   } catch (...) {
       // Обработка/генерирование исключительных ситуаций
 }
 }

// Служит как прокси вызова к ValueListHandler
// для получения списка  проектов. См. паттерн Value List Handler
 public Collection getProjectsList(Date start,
 Date end) throws PSAException {
   try {
     ProjectListHandlerHome projectVLHHome =
       (ProjectVLHHome)
         ServiceLocator.getInstance().getHome(
           "ProjectListHandler",
           ProjectVLHHome.class);
  // сессионный компонент Value List Handler
     ProjectListHandler projectListHandler =
       projectVLHHome.create();
```

```
      return projectListHandler.getProjects(
                    start, end);
    } catch (...) {
        // Обработка/генерирование исключительных ситуаций
  }
  }

  ...

  public void ejbActivate() {
    ...
  }

  public void ejbPassivate() {
    context = null;
  }

  public void setSessionContext(SessionContext ctx) {
       this.context = ctx;
  }

  public void ejbRemove() {
    ...
  }
}
```

Удаленный интерфейс для Session Facade приведен в примере 8.16.

Пример 8.16 Реализация Session Facade - удаленный интерфейс

```
package corepatterns.apps.psa.ejb;

import java.rmi.RemoteException;
import javax.ejb.*;
import corepatterns.apps.psa.core.*;
```

// Обратите внимание: для краткости детали try/catch не показаны.

```
public interface ProjectResourceManager
  extends EJBObject {

  public resetEntities(String resourceId,
  String projectId, ...)
  throws RemoteException, ResourceException ;

  public void assignResourceToProject(int numHours)
  throws RemoteException, ResourceException ;

  public void unassignResourceFromProject()
  throws RemoteException, ResourceException ;

  ...

  public ResourceTO getResourceData()
  throws RemoteException, ResourceException ;

  public void setResourceData(ResourceTO resource)
  throws RemoteException, ResourceException ;

  public ResourceTO createNewResource(ResourceTO resource)
```

```
  throws ResourceException ;

  public void addBlockoutTime(Collection blockoutTime)
  throws RemoteException,BlockoutTimeException ;

  public void updateBlockoutTime(Collection blockoutTime)
  throws RemoteException,BlockoutTimeException ;

  public Collection getResourceCommitments()
  throws RemoteException, ResourceException;

  public ProjectTO getProjectData()
  throws RemoteException, ProjectException ;

  public void setProjectData(ProjectTO project)
  throws RemoteException, ProjectException ;

  public ProjectTO createNewProject(ProjectTO project)
  throws RemoteException, ProjectException ;

  ...

  public ProjectCTO getProjectDetailsData()
  throws RemoteException, PSAException ;

  public Collection getProjectsList(Date start,
  Date end) throws RemoteException, PSAException ;

  ...
}
```

Домашний интерфейс для Session Facade показан в примере 8.17.

Пример 8.17 Реализация Session Facade - домашний интерфейс

```
package corepatterns.apps.psa.ejb;

import javax.ejb.EJBHome;
import java.rmi.RemoteException;
import corepatterns.apps.psa.core.ResourceException;
import javax.ejb.*;

public interface ProjectResourceManagerHome
extends EJBHome {

    public ProjectResourceManager create()
            throws RemoteException,CreateException;
    public ProjectResourceManager create(String
        resourceId, String projectId, ...)
            throws RemoteException,CreateException;
}
```

Связанные паттерны

- **Facade [GoF]**
  Session Facade базируется на паттерне Facade Design.
- **Data Access Object**
  Одной из стратегий для бизнес-компонента в паттерне Session Facade является

использование объекта DAO. Он применяется для простых приложений, спроектированных на использование сессионных компонентов и объектов DAO вместо компонентов управления данными.

- **Service Locator**
  Session Facade является объектом общего назначения, разрешающим инкапсуляцию рабочего процесса, управляя взаимодействием объектов бизнес-данных и бизнес-служб. Объекты бизнес-данных могут быть компонентами управления данными или объектами DAO, а объекты бизнес-служб могут быть сессионными компонентами и другими объектами, предоставляющими службы. Session Facade может использовать паттерн Service Locator для уменьшения сложности кода и использования преимуществ, предлагаемых паттерном Service Locator.
- **Business Delegate**
  Session Facade используется паттерном Business Delegate при обращении клиентского запроса к бизнес-службам. Business Delegate либо служит прокси-объектом, либо адаптирует клиентский запрос для Session Facade, который предоставляет запрашиваемую службу.
- **Broker [POSA1]**
  Session Facade выполняет роль брокера для отделения компонентов управления данными от их клиентов.